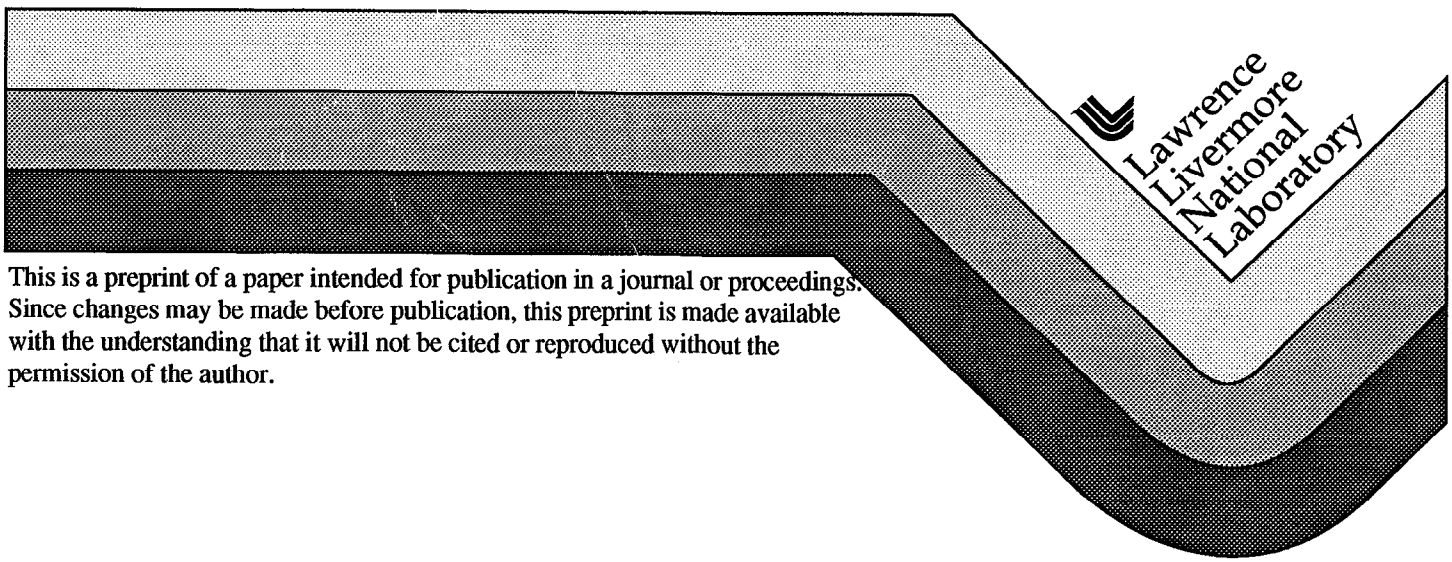
UCRL-JC-132367
Preprint

Visualizing Systems Engineering Data With Java

R. H. Barter and A. Vinzant

This paper was prepared for submittal to
9th Annual International Symposium
of the International Council on Systems Engineering
Brighton, England
June 6, 1999

November 10, 1998



Visualizing Systems Engineering Data With Java

Robert H. Barter
Lawrence Livermore National Laboratory
7000 East Ave., Livermore, CA 94550, USA

Aleta Vinzant
Lawrence Livermore National Laboratory
7000 East Ave., Livermore, CA 94550, USA

Abstract. Systems Engineers are required to deal with complex sets of data. To be useful, the data must be managed effectively, and presented in meaningful terms to a wide variety of information consumers. Two software patterns are presented as the basis for exploring the visualization of systems engineering data. The Model, View, Controller pattern defines an information management system architecture. The Entity, Relation, Attribute pattern defines the information model. MVC "Views" then form the basis for the user interface between the information consumer and the MVC "Controller"/"Model" combination.
A Java tool set is described for exploring alternative views into the underlying complex data structures encountered in systems engineering.

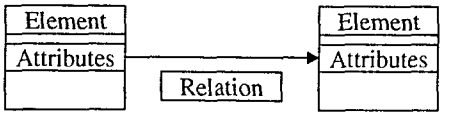
BACKGROUND

Problem Statement. Systems engineering, by its very nature, deals with enormous amounts of data. Raw, and often feral, requirements documents are typically reduced to elementary requirement statements. Requirement statements are linked, categorized, and allocated. Decision criteria is established from, and linked to, requirements. Design options are identified and design decisions are captured. Test plans, cases and procedures are developed and linked to the design and requirements. Systems Engineering Information Management (SEIM) is a common and complex problem that has provided the impetus for a lively systems engineering tool industry.
Scope. This paper addresses one aspect of SEIM, how to best present complex data sets to information consumers. This is not a commercial launch of "Yet Another Systems Engineering Tool". It is a call for interested parties to participate in exploring new strategies for presenting complex data.
Purpose. The purpose of this paper is to describe the Java tool set that we developed and to encourage others to explore novel ways of presenting complex data sets.

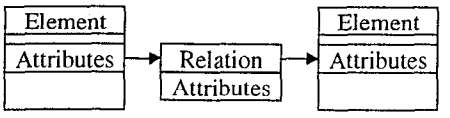
SYSTEM ARCHITECTURE

Visualization of complex data requires a software tool. The high level architecture of the tool has a profound impact on its final capabilities. There are many ways to partition software, one architecture developed at Xerox PARC in the late 1970s is described by the Model, View, Controller (MVC) pattern [Wolf]. The pattern separates the functionality of maintaining data, controlling operations on the data, and presenting data to the end user. The pattern has found wide use in applications that require human interaction with large amounts of data.
The MVC pattern was chosen for the visualization tool set because it allowed us to focus on the "View" portion of the problem. Data model and controller software was developed only to the degree necessary to support the exploration of the views. The MVC pattern has the advantage that multiple views can play off of the same data. The system "behavior" can be changed by modifying the controller software.
Model. The model that we chose to implement is the Element, Relation, Attribute (ERA) model that is common to many systems engineering tools. A minimum ERA model as shown in Figure 1a consists of elements that are linked to each other via relations. The elements contain attributes. Relations can be either unidirectional or bi-directional.
It has been suggested [include ref] that relations should include attributes as shown in Figure 1b. We have adopted the suggestion for our model and have additionally allowed for fan-in and fan-out relations as shown in Figure 1c.
The use of directional fan-in/fan-out supports simple unidirectional relations as its simplest case, and supports bi-directional relationships by linking both elements to both sides of the relation. Fan-in/fan-out also supports "commodity" relations in which it is not important to distinguish between the elements that contribute to the input side of the relation or

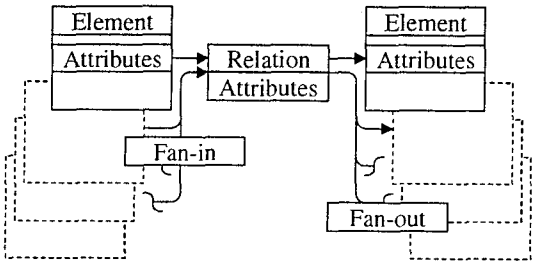
between the receiving elements on the output side of the relation.



(a) Simple



(b) Relation Attributes



(c) Fan-in Fan-out

Figure 1.
Element Relation Attributes

It is important to keep in mind that the model in an MVC implementation can be extremely general. Specialized behavior is, generally, implemented in the Controller portion of the MVC. As an example, elements in our model do not require unique identifiers beyond the object identifiers managed by Java. The Controller generates and imposes unique identifies for the benefit of the user.

Attributes are associated with elements and relations. Within a single element or relation, all of the attributes must have unique names. The model imposes no other restriction on the use of attributes. At the model level, it is perfectly acceptable for one element to have an attribute called *Function* and another element to have an attribute called *Functionality* (not a desirable situation). Again, rational, predictable behavior is the responsibility of the controller.

We have chosen to implement the model as a set of local, memory resident objects. A more sophisticated application could choose to store its data in a database or any commercial product that exposes an Application Programming Interface or API to the model.

View. A detailed discussion of the views implemented in the system is left for later in this paper. It is important to note, here, that multiple views can look at the same data. In general, views receive references to the model data by way of the controller. Once a reference is obtained, the view reads the data directly from the model. Modification to the model data is done through the controller.

Controller. The controller establishes order out of chaos and imparts a specific behavior to the application. For our purposes, the controller is fairly loose in what it allows into the model. An organization wishing to use such an MVC application might tailor the controller to reflect its engineering policy, procedures and processes.

As an example, a configuration management policy might dictate that all elements and relations be tracked for changes. Procedures might dictate that every time an element is changed, the element will be tagged with a time stamp, the nature of the change, and the person that made the change. The controller would then be required to implement a process of obtaining the user's identification (with the help of a View), and assigning the user's ID, time of change and nature of change to attributes in the Model.

The controller is also responsible for importing data into the application and exporting data from the application. Printing can be implemented in either the controller or the views. The choice will depend upon how closely the printed data is required to match a specific view of that data.

DESIGN OPTIONS

Criteria. Jumping on the bandwagon. In order to investigate the visualization techniques we had in mind, we wanted a development language that supported a wide range of platforms, had a large user community, and included a visual tool set.

Block structured and object oriented programming are two alternative paradigms for software development. Some languages are inherently object oriented. Great holy wars have been fought over subtle distinctions between one language and another.

We considered C, C++, Visual Basic, Smalltalk, Java, and Python. C and C++ have the largest user community. C++ is object oriented while C is not. Visual Basic is proprietary and object oriented. Smalltalk and Java are inherently object oriented. Python is an interesting object oriented, interpreted language with a strong following.

DESIGN RATIONALE

Object orientation was chosen over a block structured paradigm because of the potential for a group of views to inherit characteristics from a basic or parent view. Additionally, elements and relations of the ERA model are best implemented as objects.

Java was chosen because it is a simple, object oriented language that has the greatest potential for working across different platforms.

EXPLORING COMPLEX DATASETS

Infrastructure. The MVC pattern places data in the Model, behavior in the Controller, and the user interface in the Views.

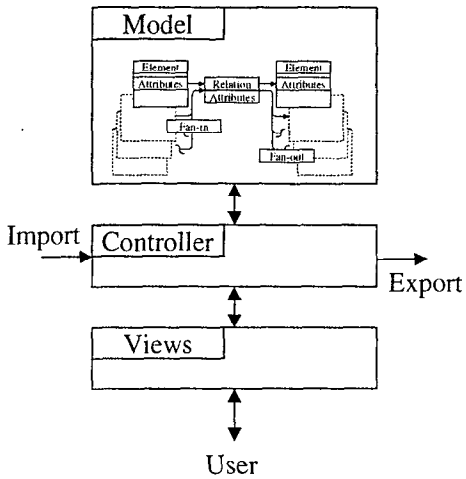
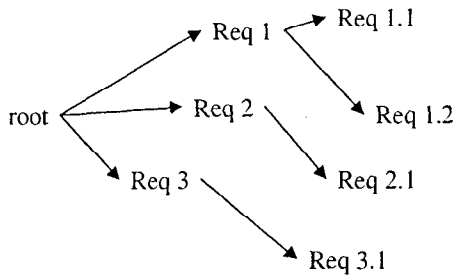


Figure 2.
Model, View, Controller, Pattern

ERA data can be categorized and viewed in many different ways. The manner in which data is viewed can be forced upon the data (and hence the user) or it can be a natural view of some intrinsic characteristic of the data.

As an example, a hierarchical requirements document in which the customer grouped subordinate requirements under superior requirements should be viewed as hierarchical data as in Figure 3a. To suppress the hierarchy to a linear list of requirements as in Figure 3b is to throw away some of the information contained in the original document. In a similar fashion, Figure 3c shows the same requirements as a graph that, while interesting, probably does not add to the user's comprehension of the information in the document.

Req 1	Req 1
Req 1.1	Req 1.1
Req 1.2	Req 1.2
Req 2	Req 2
Req 2.1	Req 2.1
Req 3	Req 3
Req 3.1	Req 3.1
(a) Original	(b) Flattened List



(c) Graph

Figure 3.
Three Views of the Same Information

It is useful to categorize data as either linear, hierarchical, intra-relational, and inter-relational. The categories are important because each category implies a different class of views.

Linear List. List elements are strongly related by some characteristic of interest. The strongly related elements do not have a superior/subordinate (or parent/child) relation with respect to the characteristic of interest (i.e. element type).

Hierarchical. Hierarchical elements are strongly related by some characteristic of interest. The strongly related elements have a parent/child relation. Functional and physical hierarchies are both examples of this category.

Intra-relational. As with hierarchical elements, intra-relational elements are strongly related by some characteristic of interest. Intra-relational elements, however, lack the parent/child relationship that characterizes hierarchical relationships. Functional flow block diagrams fall into this category.

Inter-relational. Inter-relational elements have a strong relationship between elements of differing characteristics. Functions that map onto components are inter-related.

VISUALIZING FRAT

As an example of data relationship categories, consider the methodology and data structures in "The Engineering of Complex Systems" [Mar and Morais]. In brief, the methodology categorizes system engineering information as Functions, Requirements, Answers, and Tests (FRAT). We have recently used the methodology to engineer an Integrated Safety Management System [Barter and Morais] in which we built up the following relationships.

The functionality (Functions) and design (Answers) information was captured hierarchically as seen in Figure 4a. At any one level of the functional hierarchy, the intra-relationships between the Functions were captured as functional flows as seen in Figure 4b. Figure 4c shows how inter-relationships were used to map between the Functions and their corresponding performance Requirements, between Functions and Answers, and between Functions, performance Requirements, design Answers and Tests.

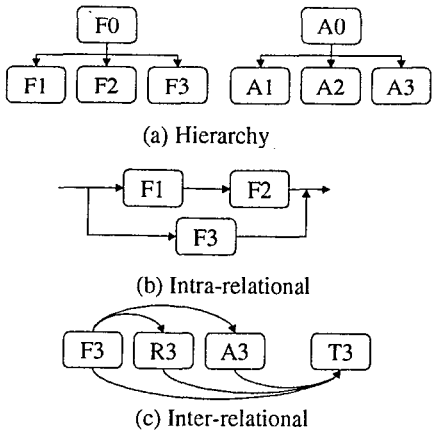


Figure 4.
FRAT Relationships

JAVA IMPLEMENTATION

The first view that we implemented in Java was a utility tool to define/explore elements and their relationships.

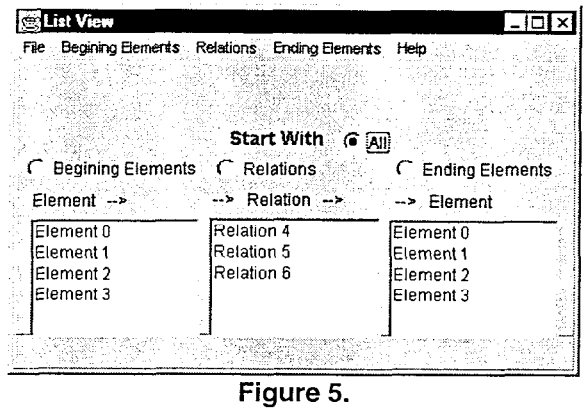


Figure 5.
Element/Relation/Element Editor

Figure 5, above, shows all of the elements and relations in the system without regard to which relations match to which elements. From this window it is possible to add or delete elements and relations.

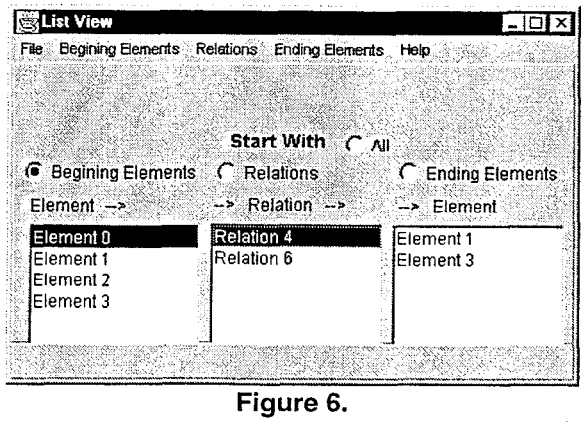


Figure 6.
Element to Relation to Element Mapping

Figure 6, above, shows that Element 0 has two Relations (4 and 5), and that Element 0 is related to Elements 1 and 3 by way of Relation 4.

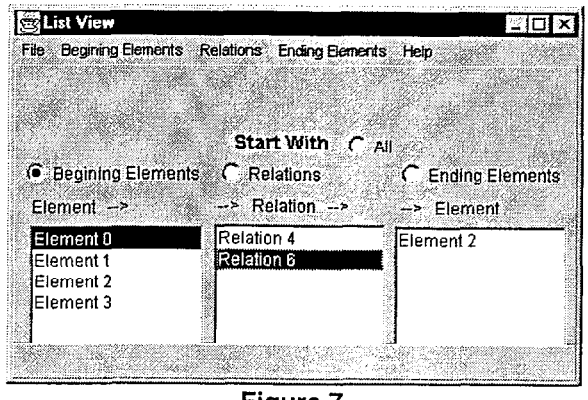


Figure 7.
Relation 6

Figure 7, above, shows that Element 0 is also related Element 2 by way of Relation 6.

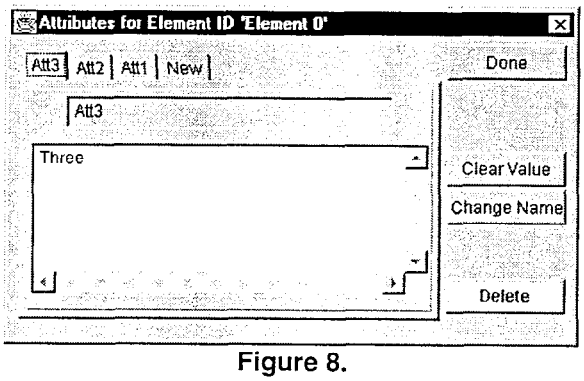


Figure 8.
Attributes for Element 0

By opening an element, it is possible to edit the attributes for that element. Figure 8, above, shows that Element 0 has three attributes named Att1, through Att3. The window uses a folder tab metaphor for selecting attributes of interest.

Once we had the ability to define and manipulate elements at a primitive level, we needed a way to reduce the number of potential elements down to a manageable set. Figure 9, below, shows one possible user interface for querying the engineering data. In this view, the user is presented with a wiring diagram metaphor. Starting on the left side of the window, the user has selected two Element Lists that, by convention, contain all of the elements in the system. The Select icons indicate that a selection criteria is being used to select a subset of all of the elements.

An example of a selection might be to select all of those elements on the input side that have attributes "Element Type" that equal "Requirement" and present that subset of elements on the output side of the Select icon.

The output of the top Select icon is fed into a Traverse icon to follow some relationship to a target

set of elements. The resulting list of elements are then available at the output of the Traverse.

The Add icon takes the two lists at its input and combines them into an output list that is presented to the final Element List. Note that Element Lists could have been placed at intermediate points in the diagram as debugging aids or to simply show intermediate results of the query.

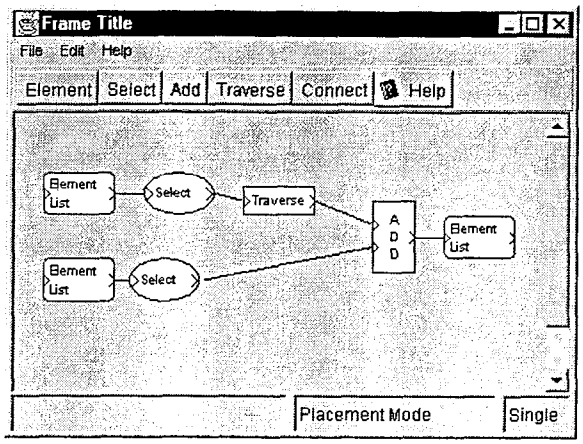


Figure 9.
Query Builder

Figure 10, below, shows an Element List Window that is obtained by double-clicking on one of the Element List icons in Figure 9, above. The Element List is rendered as a hierarchy on the left (in this case as a linear list since there is no hierarchy) and a spreadsheet on the right to show the attributes of each element. Note that in this example the elements in the system are the query elements from Figure 9, above. In general, the elements would be system engineering elements such as Requirements, Functions, Components, Sub-assemblies and the like. Hierarchical relationships would be shown as appropriate.

The column titles in Figure 10 are active buttons that, when clicked, produce a popup list of all of the possible attribute names. Selecting a new attribute name will cause that attribute to be displayed.

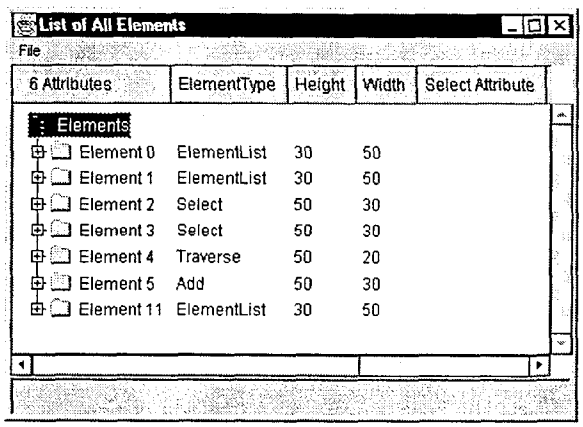


Figure 10.
Element List

COMPOSIT VIEWS

The real strength of Java comes from its ability to build complex views in a relatively short amount of time (one to two days per view is reasonable for an experienced Java programmer).

Using the Query Builder as a starting point, a multi-list view can be constructed for the FRAT data mentioned earlier. Figure 11, below shows one possible view with FRA & T hierarchy at the top and panels below to show the F to R, F to A, and F, R A to T relationships.

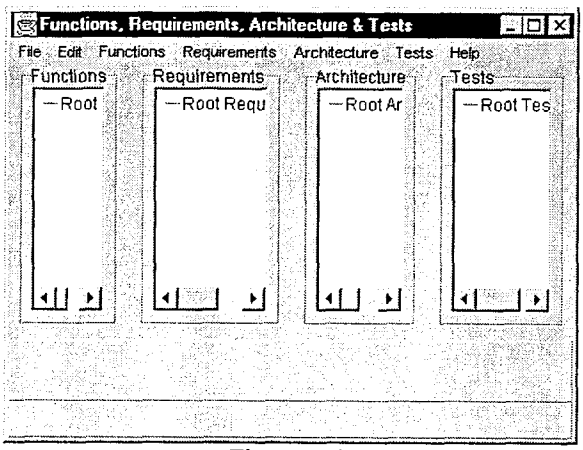


Figure 11.
FRAT View

Of a more general nature, there are two types of N^2 Diagrams that make interesting views. One type of N^2 Diagram captures the intra-relationships for a chosen set of element types. Figure 12, below, shows such a diagram. With Java's object oriented programming model and the ability to trigger events when the mouse pointer enters and leaves a cell, it is relatively easy to construct a "lively" view in which fields of additional information come and go as the mouse pointer sweeps over the window.

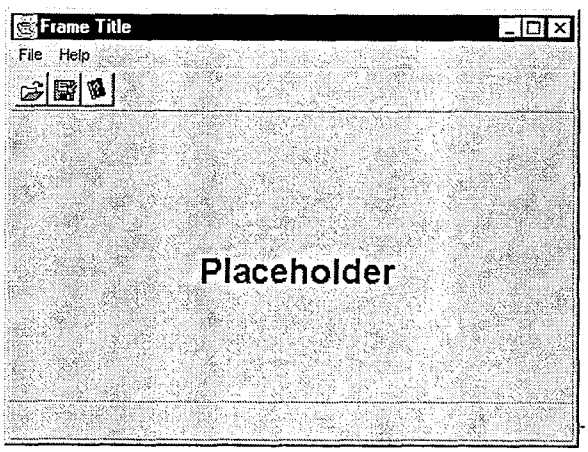


Figure 12.
N^2 Diagram For Selected Elements

Another type of N^2 Diagram documents the relationships between element types. Figure 13, below shows such a diagram. Note that the element types are shown in the diagonal elements while the relationship names are shown in the off diagonal elements.

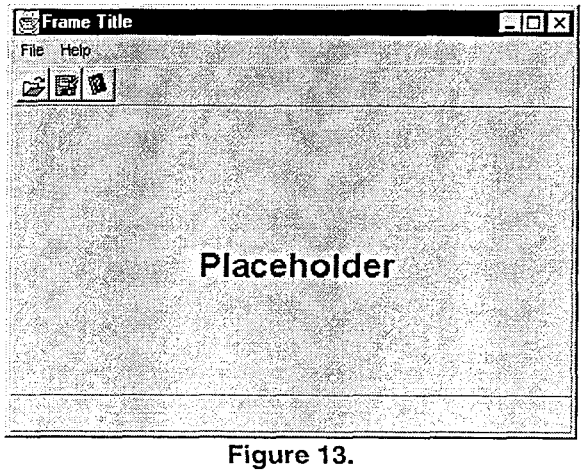


Figure 13.
N^2 Diagram For Element Relationships

Anyone up for a three-dimensional, virtual reality fly-through of an N^3 Diagram?

AVAILABILITY AND FUTURE DIRECTIONS

The software presented here is not intended for commercial use. It is, however, available for collaborative development. We are interested in a collaborative effort to develop a library of user interfaces for the benefit of the systems engineering community. Please contact the authors at the address below if you are interested in participating in the effort.

CONCLUSION

Java is an exciting language for exploring graphical user interfaces into complex systems engineering data. The object oriented nature of the software makes it an ideal vehicle for developing a library of user interfaces.

The use of a Model/View/Controller pattern and the Element/Relationship/Attribute data model make the software adaptable to differing engineering environments. While not a commercial product, the software can be useful for educational and research purposes by those willing to invest the time to become familiar with the software's underlying structure.

References:

Barter, R. H. and Morais, B. G., "*Systems Engineering Applied to Integrated safety Management for High Consequence Facilities*", Submitted for PROCEEDINGS OF THE NINETH ANNUAL INTERNATIONAL SYMPOSIUM OF THE INTERNATIONAL COUNCILE ON SYSTEMS ENGINEERING, 1999

Mar, B. W. and Morais, B. G., "*The Engineering of Complex Systems*", SYNERGISTIC APPLICTIONS, Inc., 1997

Purves, B. "*Information Models as a Prerequisite to Software Tool Interoperability*", INCOSE Insight, Vol.1 Issue 3, INCOSE, 1998

Wolf, K. and Liu, C. *"New Clients with Old Servers: A Pattern Language for Client/Server Frameworks"*, "*Pattern Languages of Program Design*", ADDISON-WESLEY PUBLISHING COMPANY, Inc., 1995